模型评价与模型改善

CS245　数据科学基础　陆朝俊
叶泽林 515030910468

1　问题描述

给定一个机器学习问题（如回归或分类），。通过组合模型的方式，可以提高模型的效果。这次作业中，我将在自己建立的分类数据集上探索决策树模型单独工作以及组合工作时的性能。

2　解决方案[1]

2.1　数据集的建立及模型的选择

由于现有的中小型数据集存在一些难以满足实验要求的属性，我选择自己建立一个分类数据集 D，之后的所有实验均基于此数据集上进行。数据集 D 包含 500 个样本，30 个特征（存在冗余和重复的特征），为更好地看出模型性能的区别，我在 D 中添加了部分噪声。关于数据集 D 详细的设定参数可参考附录 **??**。

解决分类问题的机器学习模型主要有逻辑回归、决策树以及 KNN，在这次作业中，我选择决策树作为探索的模型。

2.2　探索模型单独工作的性能

选定数据集和模型后，我首先探索单独的决策树模型在数据集 D 上的性能，探索方法如下：

1. **直接划分训练集和测试集**

   在一般应用中，对模型性能最朴素的评价方法即为：将数据集划分为训练集和测试集，在训练集上训练模型后，根据模型在测试集上的表现评价模型性能。在这次作业中，我首先采用这个方法对模型的性能进行一个粗略的估计。

2. **分层交叉验证**

   从理论上说，仅凭在一个训练集 D_1 上训练出的模型 M_1 无法代表模型 M（在我的实验中为决策树模型）解决这个问题的效果。分层交叉验证正好解决了这一问题，其步骤大致如下：

   (a) 将数据随机等分为 k 个不相交子集 $D_1, D_2, ..., D_k$（每个子集的类分布与初始数据近似相同）；

   (b) 总共执行 k 次训练与测试，在第 i 折时，使用 D_i 作为测试集, 其他子集作为训练集；

   (c) 模型的性能由 k 次迭代的平均效果决定。

   分层交叉验证相当于在同一个数据集上进行了多次训练集-测试集划分。和简单的训练集-测试集划分相比，能更准确评估模型来自样本的误差。

3. **网格搜索**

   网格搜索是通过蛮力试验不同参数组合，从而选择最佳模型的方法。理论上来说，网格搜索与分层交叉验证相结合，可以找出模型解决给定问题最合适的参数，但这一过程需要花费大量的时间和计算资源，因此这次作业中我仅实现了一小部分参数空间上的网格搜索。

[1]本次作业的所有代码实现可参见附录 A.3

2.3 探索模型组合工作的性能

从一系列模型 $M_1, M_2, ..., M_k$ 创建组合模型 M^*，可以有效提高原模型的效果。主要的方法分为 bagging 和 boosting 两种。Bagging 方法使用并行思想，M^* 通过整合各原模型的结果得到输出；Boosting 方法则采用串行思想，利用模型 M_{i-1} 的训练结果训练模型 M_i，直到 M_k。

3 结果展示

3.1 模型单独工作的性能

3.1.1 直接划分训练集和测试集

3.1.2 交叉验证

3.1.3 网格搜索

3.1.4 结果分析

3.2 模型组合工作的性能

3.2.1 bagging

3.2.2 boosting

3.2.3 结果分析

A 附录

A.1 Boston 数据集特征信息

表 1: Boston 数据集特征信息

编号	特征名	特征含义
1	CRIM	城镇人均犯罪率
2	ZN	住宅用地超过 25000 sq.ft. 的比例
3	INDUS	城镇非零售商用土地的比例
4	CHAS	查理斯河空变量（如果边界是河流，则为 1；否则为 0）
5	NOX	一氧化氮浓度
6	RM	住宅平均房间数
7	AGE	1940 年之前建成的自用房屋比例
8	DIS	到波士顿五个中心区域的加权距离
9	RAD	辐射性公路的接近指数
10	TAX	每 10000 美元的全值财产税率
11	PTRATIO	城镇师生比例
12	B	1000(Bk-0.63)2，其中 Bk 指代城镇中黑人的比例
13	LSTAT	人口中地位低下者的比例

A.2 PCA 算法在主成分个数为 3 时的进一步可视化

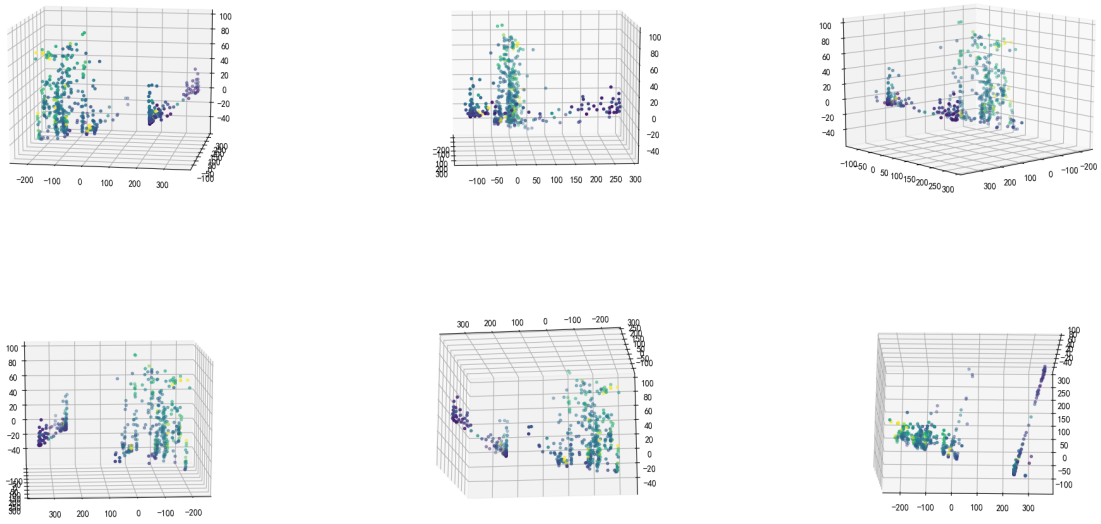

图 1: PCA 效果进一步可视化（主成分个数为 3 时）

A.3 主要代码

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']     # display Chinese
plt.rcParams['axes.unicode_minus'] = False       # display minus sign
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

import os
import time

class bostonAnalyzer(object):
    def __init__(self):
        # load dataset
        self.dataset = datasets.load_boston()
        self.data = self.dataset.data
        self.target = self.dataset.target

        # for plot
        self.var = []
        self.t = []

        if not os.path.exists('report/img'):
            os.makedirs('report/img')

    def run(self, vis=False):
        '''
        Run PCA for 1-13 dimensions
        :param vis: True: plot dim 1-3; False: not plot
        '''

        self.var.clear()
        self.t.clear()

        with open('report/result.txt', 'w') as f:
            for i in range(13):
                t = time.time()
                pca_op = PCA(n_components=i)
                pca_res = pca_op.fit_transform(self.data)
                t = time.time() - t

                self.var.append(pca_op.explained_variance_ratio_.sum()
                    )
                self.t.append(t)

                # write log
                f.write('###### Dimension %d ######\n' % i)
                f.write(str(pca_res.shape) + '\n')
```

```
47                  f.write(str(pca_op.explained_variance_ratio_) + '\n')
48                  f.write(str(pca_op.explained_variance_ratio_.sum()) +
                        '\n')
49                  f.write(str(t) + '\n\n')
50
51                  # visualize for debug & plot
52                  if vis:
53                      if i == 1:
54                          # plot 1 dimension
55                          plt.scatter(pca_res[:, 0], pca_res[:, 0], s
                                =14, c=self.target)
56                          plt.savefig('report/img/pca-%d' % i)
57                          plt.show()
58                      elif i == 2:
59                          # plot 2 dimensions
60                          plt.scatter(pca_res[:,0], pca_res[:,1], s=8, c
                                =self.target)
61                          plt.savefig('report/img/pca-%d' % i)
62                          plt.show()
63                      elif i == 3:
64                          # plot 3 dimensions
65                          ax = plt.subplot(projection='3d')
66                          ax.scatter(pca_res[:, 0], pca_res[:, 1],
                                pca_res[:, 2], s=8, c=self.target)
67                          plt.savefig('report/img/pca-%d' % i)
68                          plt.show()
69
70      def show(self):
71          '''
72          Show the basic info of Boston dataset & plot k-lines
73          '''
74          print(self.data.shape)
75          print(self.target.shape)
76          self._k_line_radio()
77          self._k_line_time()
78
79      def _k_line_radio(self):
80          '''
81          Plot k-line of variant radio
82          '''
83          x = list(range(len(self.var)))
84          plt.scatter(x, self.var, s=14, c='r')
85          plt.plot(x, self.var)
86          plt.xlabel('主成分个数')
87          plt.ylabel('降维后各特征方差比例之和')
88          plt.savefig('report/img/kline-radio')
89          plt.show()
90
```

```
91      def _k_line_time(self):
92          '''
93          Plot k-line of time
94          '''
95          x = list(range(len(self.t)))
96          plt.scatter(x, self.t, s=14, c='r')
97          plt.plot(x, self.t)
98          plt.xlabel('主成分个数')
99          plt.ylabel('PCA算法消耗时间(s)')
100         plt.savefig('report/img/kline-time')
101         plt.show()
102
103 if __name__ == '__main__':
104     bA = bostonAnalyzer()
105     bA.run(True)
106     bA.show()
```